FILESORT
User's Guide

Sharp APL/370
Sort Facility
(FILESORT)

I. P. Sharp
Associates

SHARP APL/370
Release 19

APL Systems Development Group
I. P. Sharp Associates Limited
2 First Canadian Place, Suite 1900
Toronto, Ontario, Canada.
M5X 1E3

Telephone 416 364-5361
Telex 06-22259 I P SHARP TOR

PREFACE

This book describes how to use the SHARP APL/370 Sort Facility (FILESORT) to specify and submit a request to sort a SHARP APL file. FILESORT uses the MVS Sort/Merge utility.

This book replaces SHARP APL Technical Note 10 "SORTREQ".

Printed in Canada
March 1987
Publication Code: 0133 8703 E4

CONTENTS

INTRODUCTION, 1
SUBMITTING A FILESORT REQUEST, 2
  The Left Argument, 2
  The Right Argument, 5
  The Result, 6
INQUIRING ABOUT REQUEST STATUS, 7
  The *SORTINQ* Function, 7
  The *SORTCODE* Function, 7
  The *SORTMINE* Function, 10
WITHDRAWING A SORT REQUEST, 10
FILESORT INPUT FILES, 10
SORTING EXAMPLES, 11
INDEX, 15

INTRODUCTION

Use the SHARP APL/370 sort facility (FILESORT) to sort files according to your specifications. Only file components that are of the same data type and that meet FILESORT's shape requirements can be processed. See "FILESORT Input Files" for type and shape requirements to ensure that the file you want to sort is within the scope of this product.

FILESORT uses the MVS Sort/Merge utility. All sorts are run as background tasks by user 1000 -- 1000 is the user number of the batch program that executes your job.

Use the functions in workspace 1 *FILESORT* to: specify and submit a request to sort a SHARP APL file; inquire about a previously submitted request; or to withdraw a previously submitted request. Specify and submit a FILESORT request with the *SORTREQ* function.

SUBMITTING A FILESORT REQUEST

Use the *SORTREQ* function in workspace 1 *FILESORT* to specify and submit a FILESORT request:

requestnumber←sortingspecifications *SORTREQ* tienumber

The result of the *SORTREQ* function is a **request number**. The left argument to *SORTREQ* is a character vector of **sorting specifications** and the right argument is the **tie number** of the file that you want to sort.

The Left Argument

The left argument to *SORTREQ* is a character vector of sorting specifications that comprise 13 parameters; only one of these parameters -- *OUTFILE* -- is not optional. To specify the parameters, separate each with a blank and enclose the complete vector in quotes. For example:

```
'OUTFILE=INVENTORY INCOMPA=9 INCOMPZ=24' SORTREQ ⍵
```

Sorting specifications can be specified in any order. The parameters are:

OUTFILE=**filename**
identifies the filename of the output file to which user 1000 should append the sorted components. This is the only parameter that you **must** specify. If you specify a filename without a library number, FILESORT prefixes it with your library number. For example, *OUTFILE*=*SORT*1 becomes *OUTFILE*=1234567 *SORT*1 where 1234567 is your user number.

If you specify a filename that does not already exist, FILESORT creates that file only if it is to be in your library; it will not create it if you prefix the name with someone else's library number. If you specify an existing file whose library number is not the same as your own, the sort is done only if you have suitable access to that file.

User 1000 must have access to any output file you specify. *SORTREQ* sets access for user 1000 to your output file unless you specify a locked file. If you use a locked file, see the *OUTLOCK* parameter below or "The right argument" (column 4) for more information.

The value of *OUTFILE* must be different from the input filename. See "Sorting examples" for an illustration of how *OUTFILE* is used.

INFILE=**filename**
is catenated to the end of your sorting specifications by the *SORTREQ* function; you do not have to specify it. It is the name of the file tied to the number you specify in the right argument.

User 1000 must have access to any input file you specify. *SORTREQ* sets access for user 1000 to your input file unless you specify a locked file; if you use a locked file, see the *INLOCK* parameter below or "The Right Argument" (column 4) for more information.

INCR=n
specifies the increment between components to be sorted. For example, if you specify *INCR*=2, then every second component is sorted. If you omit a value for *INCR*, 1 is used, causing every component to be sorted. For an illustration of the use of *INCR*, see Example 4 in "Sorting Examples".

OUTROWS=n
specifies the number of rows in each component of your output file. For example, specifying *OUTROWS*=100 causes your output file to have one hundred rows in each component. If you omit *OUTROWS*, then the value of *INROWS* is used. If *INROWS* is not specified, then FILESORT uses the number of rows in the first component of the input file. For an illustration of the use of *OUTROWS*, see Example 3 in "Sorting Examples".

INROWS=n
specifies the maximum number of rows in any one component of the input file. Although FILESORT no longer requires this information, the parameter has been retained to enable earlier applications that use FILESORT to work.

INLOCK=n
specifies the pass number that user 1000 needs to access the input file. For example:

```
INLOCK=789
```

If you omit *INLOCK*, 0 is used. Another way to specify the *INLOCK* parameter is to enter the pass number in the fourth column of the first row in the right argument; this causes *SORTREQ* to catenate *INLOCK* to the front of your sorting specifications.

OUTLOCK=n
specifies the pass number that user 1000 needs to access the output file. For example:

```
OUTLOCK=456
```

If you omit *OUTLOCK*, 0 is used. Another way to specify the *OUTLOCK* parameter is to enter the pass number in the fourth column of the second row in the right argument; this causes *SORTREQ* to catenate *OUTLOCK* to the front of your sorting specifications.

INCOMPA=n
specifies the component number of the input file from which sorting should start. For example, *INCOMPA*=6 causes sorting to start at component 6 of the input file. The default value for *INCOMPA* is the 1st component of the input file.

INCOMPZ=n
specifies the component number of the input file where sorting should stop. For example, *INCOMPZ*=26 causes sorting to stop before processing component 26 of the input file. The default value for *INCOMPZ* is one more than the last component number of the input file. See Example 2 in "Sorting Examples" for an illustration of the use of *INCOMPZ*.

INTIE=intie
specifies whether or not you want user 1000 to have exclusive use of the input file during sorting. Its value can be *SHARE* or *FULL*. *FULL* indicates that user 1000 is to have exclusive use of the input file during processing; *SHARE* means that others, including yourself, also can use the file during processing. If you omit a value for *INTIE*, *SHARE* is used.

OUTTIE=outtie
specifies whether or not you want user 1000 to have exclusive use of the output file during processing. Its value can be *SHARE* or *FULL*. *FULL* indicates that you want user 1000 to have exclusive use of the output file during processing; *SHARE* means that other users also can use the output file. If you omit a value for *OUTTIE*, *SHARE* is used.

SEQ=seq
specifies the collating sequence to be used when sorting character (not numeric) data. Any repeated character ends the string. For example, if you want *B*, *P* and *E* to be sorted before anything else, specify:

```
SEQ=BPEE
```

This causes *B*, *P* and *E* to be sorted before any other characters, which are sorted in the same sequence as the characters in *⎕AV*. For an illustration of the use of *SEQ*, see Examples 3 and 4 in "Sorting Examples".

OUTDROP=outdrop
specifies whether an existing output file is to have all of its components dropped before beginning to append the sorted components. If you are dropping the components, you have to indicate whether they should be dropped from the beginning or the end of the file.

OUTDROP=NONE indicates that you don't want any components dropped; *OUTDROP=PLUS* indicates that you want components dropped from the beginning of the file; and *OUTDROP=MINUS* indicates that you want components dropped from the end of the file. If components are dropped from the beginning, the first component that is appended is numbered one greater than the last component that was dropped. If components are dropped from the end, the first component number remains the same. If you omit a value for *OUTDROP*, *MINUS* is used.

KEYS=(n m x...)
specifies the location of the keys to be used to sort the file and whether the sorting is to ascend or descend on each key. To specify *KEYS*, enter up to a maximum of 20 triplets of **n m x** in parentheses, where **n** and **m** are numbers and **x** is one of *A*, *AB*, *AS*, *D*, *DB*, or *DS*. **n** is the number (origin 1) of the column that starts each key; **m** is the number of contiguous columns that form each key; *A* or *AS* indicates that sorting is to ascend on each key, *D* or *DS* indicates that sorting is to descend on each key--suffixing *A* or *D* with a *B* indicates that a new output component is to be started when the key changes.

For example:

```
KEYS=(11 18 A)
```

In this example, 11 is the column that starts each key, 18 is the number of contiguous columns that form each key, and *A* indicates that sorting is to ascend on each key.

The first triplet you enter is the major key and any following triplets are minor keys. For example:

```
KEYS=(11 18 A 4 6 D)
```

In this example, `11 18 A` is the major key and `4 6 D` is the minor key.

If you suffix *A* or *D* with *B*, FILESORT starts a new component with each change of key.

If *KEYS* is omitted, then the file is sorted ascending on the first 256 bytes of character data of the first subarray, or on the first 12 columns of numeric data in the first subarray. For an illustration of the use of *KEYS*, see "Sorting examples".

The Right Argument

The right argument to the *SORTREQ* function can be an integer scalar, vector, or matrix. Regardless of what form it is, internally the *SORTREQ* function turns the right argument into a two-row by four-column matrix.

row 1
contains the tie number and access specifications for the input file.

row 2
contains the tie number and access specifications for the output file.

column 1
contains the tie numbers of the input and the output files. The input file tie number is the only value in the internal matrix that you must specify; *SORTREQ* supplies values for all other unspecified elements. You must specify an output file tie number when you have created your own output file and now want *SORTREQ* to set the correct user 1000 access for it.

column 2
contains the pass numbers of the input and output files. If you omit values for column two, then *SORTREQ* uses 0.

column 3
indicates to *SORTREQ* whether it should or should not set access for user 1000. 0 indicates that *SORTREQ* should set access and any positive number indicates that it shouldn't. If you omit values for column 3, then *SORTREQ* uses 0.

column 4
contains the pass numbers that user 1000 needs to access the input and output files. If you omit values for column four, then *SORTREQ* uses 0. Any values you specify in column four are catenated to the sorting specifications as *INLOCK* and *OUTLOCK* parameters.

When you specify a scalar right argument, *SORTREQ* assumes that it is the tie number of your input file and supplies values for all other matrix positions. When you specify a vector right argument, *SORTREQ* uses the first four integers only and assumes that they apply to the first row of the matrix; *SORTREQ* supplies values for the second row. The only way you can specify values for the second row of the matrix is by using a matrix right argument. For example:

```
      2 1 ⍴ 44 85
```

Internally this argument becomes:

```
44 0 0 0
85 0 0 0
```

SORTREQ unties both the input and the output file tie numbers that you specify in the right argument.

The Result

The result of the *SORTREQ* function is a request number. For example:

```
      'OUTFILE=EXAMPLE KEYS=(1 5 A)' SORTREQ 15
45666
```

The request number does not indicate the success or failure of your request. However, you can use it to inquire about your request's status or to withdraw your request. Inquire about your request's status with the *SORTINQ* function.

INQUIRING ABOUT REQUEST STATUS

Workspace 1 *FILESORT* has three functions, *SORTINQ*, *SORTCODE*, and *SORTMINE*, that you can use to obtain status information. You use *SORTINQ* to obtain a status code that represents your request's status; you use the *SORTCODE* function to display the definition of the status code returned by *SORTINQ*; and you use the *SORTMINE* function to display the request numbers of all the sort requests you have submitted in the last eight days.

The *SORTINQ* Function

Use the *SORTINQ* function in workspace 1 *FILESORT* to inquire about the status of a previously submitted sort request. For example:

```
      SORTINQ 45666
A  1234567 OUTFILE=EXAMPLE KEYS=(1 5 A)   INFILE=
     1234567 INVENTORY
```

The right argument to *SORTINQ* is the request number returned by the *SORTREQ* function. The result of *SORTINQ* is a character vector that consists of:

- a status code that indicates the status of your request. In this example, the status code is *A*.
- your user number--in this example 1234567.
- the sorting specifications you entered plus any parameters such as *INFILE*, *INLOCK*, or *OUTLOCK* that *SORTREQ* may have added.

Only a task that has the same user number as the submitter can receive status information.

Display the definition of a status code with the *SORTCODE* function.

The *SORTCODE* Function

Use the *SORTCODE* function to display the message associated with a status code. For example:

```
      SORTCODE 'A'
DONE
```

The right argument to *SORTCODE* is the status code returned by the *SORTINQ* function. The result is a message indicating the status of your request.

To display a list of the *SORTCODE* messages, enter:

```
      SORTCODE ''
```

The status codes and associated messages are:

Code: blank
Message: *UNPROCESSED*
indicates that your request is not yet processed.

Code: +
Message: *IN PROGRESS*
indicates that your request is being processed.

Code: *
Message: *WITHDRAWN*
indicates that your request has been withdrawn.

Code: *?*
Message: *REQUEST NOT YOURS*
indicates that your request number is invalid--it is not your request, or it has been dropped from the queue (requests are dropped after eight days).

Code: *A*
Message: *DONE*
indicates that your request has been processed.

Code: *B*
Message: *ERROR IN REQUEST SYNTAX*
indicates that your request contains a visible error. For example, a misspelled keyword or an output filename that is the same as the input filename.

Code: *C*
Message: *OUTFILE NOT YOURS*
indicates that you have violated the rules for forming an output filename; for example, you specified a non-existent file in someone else's library.

Code: *D*
Message: *OUTFILE CANNOT BE TIED*
indicates that user 1000 does not have sufficient access to tie the output file.

Code: *E*
Message: *INFILE CANNOT BE TIED*
indicates that user 1000 does not have sufficient access to tie the input file.

Code: *F*
is no longer issued.

Code: *G*
Message: *INFILE CANNOT BE READ*
indicates that user 1000 does not have sufficient access to read the input file.

Code: *H*
Message: *KEY ADDRESSES BEYOND* 4088
indicates that the key is too long. All parts of the key must be within the first 4088 bytes of the row. Because floating-point numbers require eight bytes each, this can begin at the 512th column of data.

Code: *I*
Message: *KEY EXCEEDS* 4088 *BYTES*
indicates that the key exceeds 4088 bytes. The sum of the lengths of all keys must not exceed 4088 bytes. This can occur without violating condition *H* because overlapping keys are permitted.

Code: *J*
Message: *KEY ADDRESSES BEYOND ROW*
indicates that you specified a key address that is greater than the number of columns in your data.

Code: *K*
is no longer issued.

Code: *L*
Message: *BAD INFILE COMPONENT*
indicates that your input file does not meet FILESORT's input file specifications.

Unsortable components include: a package, a simple array of three or more dimensions, an enclosed array that is not a vector, a vector enclosed array whose length is greater than 200, an enclosed vector that contains a subarray that is not simple, an enclosed vector that contains a subarray that is three dimensions or more, an enclosed array whose length does not match that of the first component, and an enclosed array whose subarrays do not all have the same number of rows.

Code: *M*
Message: *INFILE COMPONENT TYPE INCONSISTENT*
indicates that you are attempting to sort components that are of different data types. For example, your first component was a numeric matrix and your second was a character matrix. For more information, see "FILESORT input files".

Code: *N*
is no longer issued

Code: *O*
Message: *OUTFILE FULL OR NO USER* 1000 *ACCESS*
indicates either that user 1000 does not have append access to the output file, or that the output file is full and user 1000 does not have sufficient access to resize the file, or that it receives a *FILE RESERVATION ERROR* in attempting to do so.

Code: *P*
Message: *SORT DETECTED ERROR*
indicates that the sort utility used by user 1000 has reported an error. (Although this could indicate other errors, it usually means that the file is too large to be sorted in the work area allocated by the data centre.)

Code: *Q*
Message: *INCONSISTENT NUMBER OF COLUMNS*
indicates, in the case of simple data being sorted, that some component contains a different number of columns from the first component. In the case of enclosed data being sorted, it indicates that a subarray of some component contains a different number of columns from the corresponding subarray in the first component.

The *SORTMINE* Function

Use the *SORTMINE* function to display the request numbers of all the sort requests you have submitted in the last eight days. For example:

```
      SORTMINE
45666 45667 45678 45680
```

WITHDRAWING A SORT REQUEST

Use the monadic *SORTWITHDRAW* function to withdraw a previously submitted sort request. For example:

```
      SORTWITHDRAW 45666
* 1234567 OUTFILE=EXAMPLE KEYS=(1 5 A)   INFILE=
    1234567 INVENTORY
```

The right argument of the *SORTWITHDRAW* function is the request number returned by the *SORTREQ* function. The result of a successful withdraw request is an asterisk followed by the request itself. If the withdraw request is too late, the result begins with one of the other status codes. For a description of each of the status codes, see "Inquiring about Request Status". Only a task that has the same user number as the submitter can withdraw a request.

FILESORT INPUT FILES

All file components of a FILESORT input file must be of the same data type and must meet FILESORT's shape requirements.

Data type
All file components to be sorted must be of the same data type: they can be character matrices; numeric matrices of all numeric subtypes except complex; or vectors of enclosed arrays. They cannot be a mixture of these types.

Component shape
FILESORT does not accept arrays of rank greater than 2. If you are sorting numeric or character matrices, all components to be sorted must have the same number of columns; the number of rows in each component can be different.

If you are sorting vectors of enclosed arrays, all components to be sorted must have the same number of enclosed arrays--the

length of each vector can be no more than 200. Different arrays within the same component must have the same number of rows, but can differ in data type and in the number of columns. The nth array in all components to be sorted can differ in the number of rows, but the data type and number of columns must be the same.

SORTING EXAMPLES

The following sorting examples use input file 15 shown below:

```
      'EXAMPLE' ⎕CREATE 15
((8 5 ⍴ ⍳4)⊃ 8 10⍴'GOBBLEDGOOK ')⎕APPEND 15
((4 5 ⍴ ⍳3)⊃ 4 10⍴'ABCDEFGHIJKLMNOPQRSTUVWXYZ ')
      ⎕APPEND 15
```

The file's components look like:

```
      ⎕READ 15 1
0 1 2 3 0 GOBBLEDGOO
1 2 3 0 1 K GOBBLEDG
2 3 0 1 2 OOK GOBBLE
3 0 1 2 3 DGOOK GOBB
0 1 2 3 0 LEDGOOK GO
1 2 3 0 1 BBLEDGOOK
2 3 0 1 2 GOBBLEDGOO
3 0 1 2 3 K GOBBLEDG

      ⎕READ 15 2
0 1 2 0 1 ABCDEFGHIJ
2 0 1 2 0 KLMNOPQRST
1 2 0 1 2 UVWXYZ ABC
0 1 2 0 1 DEFGHIJKLM
```

Example 1

Sort *EXAMPLE* so that the keys located in columns 1 to 15 are sorted in descending order:

```
      'OUTFILE=SORTED KEYS=(1 15 D)' SORTREQ 15
46706
```

Tie *SORTED* to 16:

```
      'SORTED' ⎕STIE 16
```

Display *SORTED*:

```
      ⎕READ 16 1
3 0 1 2 3 K GOBBLEDG
3 0 1 2 3 DGOOK GOBB
2 3 0 1 2 OOK GOBBLE
2 3 0 1 2 GOBBLEDGOO
2 0 1 2 0 KLMNOPQRST
1 2 3 0 1 K GOBBLEDG
1 2 3 0 1 BBLEDGOOK
1 2 0 1 2 UVWXYZ ABC
```

```
      □READ 16 2
0 1 2 3 0 LEDGOOK GO
0 1 2 3 0 GOBBLEDGOO
0 1 2 0 1 DEFGHIJKLM
0 1 2 0 1 ABCDEFGHIJ
```

Example 2

Sort the first component of *EXAMPLE* so that the keys located in the first five columns are sorted in ascending order and so that a new component is started at each change of key:

```
      'OUTFILE=SORTED1 INCOMPZ=2 KEYS=(1 5 AB)'
          SORTREQ 15
46705
```

Tie *SORTED1* to 17:

```
      'SORTED1' □STIE 17
```

Display *SORTED1*:

```
      □READ 17 1
0 1 2 3 0 GOBBLEDGOO
0 1 2 3 0 LEDGOOK GO

      □READ 17 2
1 2 3 0 1 K GOBBLEDG
1 2 3 0 1 BBLEDGOOK

      □READ 17 3
2 3 0 1 2 OOK GOBBLE
2 3 0 1 2 GOBBLEDGOO

      □READ 17 4
3 0 1 2 3 DGOOK GOBB
3 0 1 2 3 K GOBBLEDG
```

Example 3

Sort *EXAMPLE* so that the keys located in column 6 and the next five contiguous columns are sorted in ascending order, and so that any keys starting with *K* and then *D* sort first. Specify that there are to be three rows in each component of the output file.

```
      'OUTFILE=SORTED2 SEQ=KDD KEYS=(6 5 A)
           OUTROWS=3' SORTREQ 15
47506
```

Tie *SORTED2* to 18:

```
      'SORTED2' □STIE 18
```

Display *SORTED2*:

```
      □READ 18 1
2 0 1 2 0 KLMNOPQRST
1 2 3 0 1 K GOBBLEDG
3 0 1 2 3 K GOBBLEDG

      □READ 18 2
0 1 2 0 1 DEFGHIJKLM
3 0 1 2 3 DGOOK GOBB
0 1 2 0 1 ABCDEFGHIJ

      □READ 18 3
1 2 3 0 1 BBLEDGOOK
0 1 2 3 0 GOBBLEDGOO
2 3 0 1 2 GOBBLEDGOO

      □READ 18 4
0 1 2 3 0 LDEGOOK GO
2 3 0 1 2 OOK GOBBLE
1 2 0 1 2 UVWXYZ ABC
```

Example 4

Sort every second component of *SORTED2* so that the keys located in column 6 and the next five contiguous columns are sorted in ascending order and so that *G* and then *K* sort first. If two or more keys in the major key are the same, sort in descending order on minor keys located in columns 1 to 5.

```
      'OUTFILE=SORTED3 INCR=2 SEQ=GKG
           KEYS=(6 5 A 1 5 D) SORTREQ 18
46809
```

Tie *SORTED3* to 19:

```
      'SORTED3' □STIE 19
```

Display *SORTED3*:

```
      □READ 19 1
2 3 0 1 2 GOBBLEDGOO
0 1 2 3 0 GOBBLEDGOO
2 0 1 2 0 KLMNOPQRST

      □READ 19 2
3 0 1 2 3 K GOBBLEDG
1 2 3 0 1 K GOBBLEDG
1 2 3 0 1 BBLEDGOOK
```

Access, full, to input/output files 4
Access, share, to input/output files 4
Arrays, requirements for input files 10-11
Collating sequence, specifying 4
Component shape. See Input files
Data type. See Input files
Enclosed arrays, requirements for sorting 10-11
FILESORT request,
  how to specify and submit 2
  how to withdraw 10
  inquiring on status 7
FILESORT, general overview 1
Full access, to input/output files (user 1000) 4
INCOMPA. See Parameters
INCOMPZ. See Parameters
INCR. See Parameters
INFILE. See Parameters
INLOCK. See Parameters
Input file,
  component shape of 1,10-11
  data type of 1,10
  access specifications 5
  component number of, where sorting should start 3
  component number of, where sorting should end 4
  exclusive use of, during sorting 4
  number of rows in component of 3
  pass number needed to access the 3,6
  tie number for 5
  user 1000 access to 3
Inquiry, of request status 7
INROWS. See Parameters
INTIE. See Parameters
KEYS. See Parameters
Library number, as prefix to output filename 2
Matrices, requirements for sorting 10
Messages, associated with status codes. See Status codes.
MINUS, setting of *OUTDROP*, 4
NONE, setting of *OUTDROP*, 4
OUTDROP. See Parameters
OUTLOCK. See Parameters
Output file,
  access specifications for 5
  dropping components in 4
  exclusive use of, during processing 4
  filename of, as differs from input filename 2
  identify the filename 2
  number of rows in component of 3
  tie number of 5
  user 1000 access to 2
OUTROWS. See Parameters
OUTTIE. See Parameters

Parameters,
INCOMPA 3
INCOMPZ 4
INCR 3
INFILE 2-3
INLOCK 3
INROWS 3
INTIE 4
KEYS 5
OUTDROP 4
OUTFILE 2
OUTLOCK 3
OUTROWS 3
OUTTIE 4
SEQ 4
Pass numbers, of input and output files 3,6
PLUS, setting of *OUTDROP* 4
□*AV*, default collating sequence 4
Request status, inquiry of 7
SEQ. See Parameters
Share access to input/output files 4
Sort keys, specifying location of 5
SORTCODE function,
messages displayed by 7-9
result of 7
right argument to 7
use of 7
Sorting,
collating sequence for character data 4
examples 11-13
specifying increments between components 3
specifying keys 5
SORTINQ function,
result of 7
right argument to 7
use for status inquiry 7
SORTMINE function, use of 10
SORTREQ function,
examples of use 11-13
left argument of 2
parameters 2-5
result of 2
right argument of 5-6
specifying parameters in 2-5
use of 2
SORTWITHDRAW function,
result of 10
right argument of 10
use of 10
Status codes, code:
A (done) 8
B (error in request syntax) 8
blank (unprocessed) 8
C (outfile not yours) 8
D (outfile cannot be tied) 8
E (infile cannot be tied) 8
F 8
G (infile cannot be tied) 8
H (key addresses beyond) 9
I (key exceeds 4088 bytes) 9

J (key addresses beyond row) 9
K 9
L (bad infile component) 9
M (infile component file inconsistent) 9
N 9
O (outfile full or no user 1000 access) 9
P (sort detected error) 9
Q (inconsistent number of columns) 10
+ (in progress) 8
* (withdrawn) 8
? (request not yours) 8
Tie numbers,
for the input file 5
for the output file 5
when to specify 5
User 1000,
access to input file 3
access to output file 2
use in sorting, overview 1
Withdraw sort request (*SORTWITHDRAW*) 10